

Patent

UNITED STATES PATENT APPLICATION

FOR

METHOD AND APPARATUS FOR AN IN-SITU VICTIM CACHE

INVENTOR:

**WILLIAM G. AULD
ZHONG-NING CAI**

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD, SEVENTH FLOOR
LOS ANGELES, CA 90025-1030
(408) 720-8598

ATTORNEY'S DOCKET NO. 42P17019

Express Mail Certificate

"Express Mail" mailing label number: EV305339536US

Date of Deposit: October 28, 2003

I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to Mail Stop Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria VA 22313-1450

Anne Collette

(Typed or printed name of person mailing paper or fee)

Anne Collette

(Signature of person mailing paper or fee)

10/28/2003

(Date signed)

METHOD AND APPARATUS FOR AN IN-SITU VICTIM CACHE

FIELD

[0001] The present disclosure relates generally to microprocessors,
5 and more specifically to microprocessors utilizing a cache hierarchy.

BACKGROUND

[0002] Modern microprocessors often use a cache hierarchy. In a hierarchy of caches, inner caches, those more closely connected to a processor, may give rapid access to instructions and data but may have
10 limited storage capacity. Outer caches, those more closely connected to system memory, may have much larger storage capacity but much larger latency for requests from the processor. No matter whether they are inner caches or outer caches, caches generally face the problem of the replacement of cache lines that are no longer needed with new
15 cache lines representing newer memory requests from the processor. When there are cache lines in an invalid state, these may be replaced with new cache lines without much system performance impact. But when no cache lines are in an invalid state, the cache control logic must pick a cache line in a valid state, called a victim cache line, and
20 overwrite it with the new cache line. This presents a system performance issue in that a cache line discarded as a victim may soon be requested again by the processor, and may need to be fetched once again.

[0003] There are several existing methods of selecting a victim cache
25 line when needed. One commonly used one is the least-recently-used (LRU) method, where the cache line selected as the victim cache line is the one which has been unused for the longest time. However, when

used in an outer cache, the LRU method may give poor results because what may be important is which cache lines have been unused by the inner caches, and the outer caches may not have access to this information. Similarly, random victim selection and first-in-first-out (FIFO) victim selection methods may have drawbacks.

- [0004]** One approach to dealing with poorly-chosen victim cache lines utilizes a small buffer called a victim cache, connected between its related cache and the refill path. The victim cache may contain the victim cache lines that were recently overwritten in the related cache.
- 10 The victim cache may be checked on a cache line miss, and if found in the victim cache the requested cache line may be swapped with another cache line in the related cache. This may reduce the latency of fetching the requested cache line from more outer caches or system memory, but at the cost of the complexity of the victim cache and its associated
- 15 logic. As the victim cache may need to ensure cache coherency, this cost may be relatively high.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

5

[0006] **Figure 1** is a schematic diagram of a cache hierarchy, according to one embodiment.

[0007] **Figures 2A and 2B** are schematic diagrams of cache lines in a cache, according to one embodiment.

10 **[0008]** **Figure 3** is a schematic diagram of messages and transactions between outer and inner caches, according to one embodiment of the present disclosure.

[0009] **Figure 4** is a schematic diagram of messages and transactions between coherent caches, according to one embodiment of the present
15 disclosure.

[0010] **Figures 5A and 5B** are flowcharts of a method of operation of an in-situ victim cache, according to one embodiment of the present disclosure.

[0011] **Figure 6** is a schematic diagram of a microprocessor system,
20 according to one embodiment of the present disclosure.

DETAILED DESCRIPTION

[0012] The following disclosure describes techniques for a cache to select and validate victim cache lines without the use of an external victim cache. In the following description, numerous specific details such as logic implementations, software module allocation, bus signaling techniques, and details of operation are set forth in order to provide a more thorough understanding of the present invention. It will be appreciated, however, by one skilled in the art that the invention may be practiced without such specific details. In other instances, control structures, gate level circuits and full software instruction sequences have not been shown in detail in order not to obscure the invention. Those of ordinary skill in the art, with the included descriptions, will be able to implement appropriate functionality without undue experimentation. The invention is disclosed in the form of a processor using caches in a linear hierarchy. However, the invention may be practiced in other configurations of caches, such as shared outer caches servicing a multi-core processor architecture.

[0013] Referring now to Figure 1, a schematic diagram of a cache hierarchy is shown, according to one embodiment. The Figure 1 processor 100 may include a front end 102 and a back end 104. The front end 120 may include a fetch/decode stage 106. The fetch/decode stage 106 may fetch (and potentially prefetch) instructions from a memory hierarchy that may in one embodiment include an L1 (level one) cache 120, an L2 (level two) cache 130, and memory 140. In the Figure 1 embodiment, L1 cache 120 and L2 cache 130 are shown as unitary caches, where a unitary cache is one that may store both

instructions and data. In other embodiments, other forms of memory hierarchy may be used.

[0014] The fetch/decode stage 106 may also decode the instructions. In one embodiment, macro-instructions may be decoded into sets of micro-operations. The front end 102 may also include some kind of inner cache to store instructions. In some embodiments, this may take the form of an L0 (level zero) instruction cache or an instruction buffer. In the Figure 1 embodiment, a trace cache 108 is used. In one embodiment, trace cache 108 may store linked sets of micro-operations called traces. Each macro-instruction may be stored in trace cache 108 in decoded form as one or more corresponding traces.

[0015] The back end 104 may receive macro-instructions or micro-operations from the front end 102 and act upon them. In one embodiment, back end 104 may include a register alias table to translate logical register addresses to physical register addresses. Back end 104 may also include a scheduler to issue instructions, including micro-operations, in an order to suit pipeline execution. A register file read/bypass stage may be included to permit reading operand values from the physical register addresses, and may also support operands from a bypass circuit within back end 104. An execution stage may execute the instructions, and may include several execution units for the execution of several instructions in parallel. After execution, the instructions may be checked for branch misprediction, exceptions, and other execution anomalies in a check stage. Instructions whose execution passes the checks may then be retired in a retirement stage, which may update the processor state as appropriate. Data required by the back end 104 may be supplied by an L0 data cache 110.

[0016] The interface 142 between the outer L2 cache 130 and memory 140 may be of various kinds. In one embodiment, interface 142 may be a system bus of some variety. In other embodiments, a point-to-point memory interface may be used to enhance performance.

5 In Figure 1, L2 cache 130 is shown supporting only one set of inner caches (L1 cache 120, trace cache 108, and L0 data cache 110). In other embodiments, processor 100 may share L2 cache 130 with other processors. In some embodiments, processor 100 may be one of several processors in a multi-core processor module. It may be noted that L2
10 cache 130 is an outer cache with respect to L1 cache 120, trace cache 108, and L0 data cache 110. L1 cache 120 is an inner cache with respect to L2 cache 130 but is also an outer cache with respect to trace cache 108 and L0 data cache 110. Trace cache 108 and L0 data cache 110 are inner caches with respect to both L2 cache 130 and L1 cache
15 120.

[0017] Referring now to Figures 2A and 2B, schematic diagrams of cache lines in a cache 200 are shown, according to one embodiment. Cache 200 may serve as, for example, the L2 cache 130 or the L1 cache 120 of Figure 1, or in many other embodiments of a cache. Cache 200
20 is shown as an 8-way set-associative cache with N sets. However, in other embodiments the cache could have other numbers of ways in each set, or could be a fully-associative cache.

[0018] The individual sets of cache 200, from set 0 204 through set N 208, each include 8 ways, labeled W0 through W7, where each way is
25 capable of storing a cache line. For example, in set 1 206, way W3 210 may include a cache line including data 230 and tag 228. The data 230 may include various data words and instructions. The tag 228 may

include a tag proper, an index, and a block offset. In other embodiments, tag 228 may include different information. In some embodiments, way W3 210 may include a core identification field 226 when cache 200 may be used with multiple cores in a multi-core processor. Core identification field 226 may be used to limit back-invalidations to lower-level caches only to those cores that may require them.

[0019] In one embodiment, way W3 210 may include cache coherency bits 222 to indicate the cache coherency status of the cache line contained therein. In embodiments using a protocol that uses the modified/exclusive/shared/invalid (MESI) states, cache coherency bits 222 may encode these states. In other embodiments, other states could be encoded in cache coherency bits 222. The validity of the cache line contained in way W3 210 may be indicated by a valid flag associated with way W3 210. In one embodiment, a decoded valid bit 220 may be made true to indicate a valid cache line and made false to indicate an invalid cache line. In other embodiments, the cache coherency bits 222 may be used as an encoded valid flag. For example, in one embodiment when cache coherency bits 222 include a pattern indicating an M, an E, or an S state, the valid flag may be considered true to indicate a valid cache line. When the cache coherency bits 222 include a pattern indicating an I state, the valid flag may be considered false to indicate an invalid cache line.

[0020] Way W3 210 may also include a victim flag. In one embodiment, a decoded victim bit 224 may be made true to indicate a victim cache line and made false to indicate a non-victim cache line. In other embodiments, a set of bits associated with each set may indicate

in encoded form which way of the set may include a victim cache line. The control of the victim flag may be accomplished by the cache control logic 202. In one embodiment, cache control logic 202 may always make true a victim flag within each set, and the way referred to by the victim flag may be selected using a well-known victim selection method such as the least-recently-used (LRU), random, or first-in-first-out (FIFO) methods. In another embodiment, cache control logic 202 will monitor the valid flag of each way in a set, and only make true a victim flag when none of the valid flags is false. When a replacement cache line is needed for a given set within cache 200, the cache control logic 202 may first determine whether any of the ways has its valid flag false. If so, one of those ways may be overwritten with the replacement cache line. If not, then the way whose victim flag is true may be overwritten by the replacement cache. However, the mere making true of a victim flag of a way does not mean that the cache line therein will definitely be evicted or overwritten: it should be considered an indicator that that cache line is a potential and pending victim cache line.

[0021] Whenever a victim flag is made true, the cache control logic 202 may send a back-invalidate signal to any inner caches of cache 200. In one embodiment, the back-invalidate signal may be necessary in an inclusive cache hierarchy. However, the cache control logic 202 will send the back-invalidate signal without waiting for an actual eviction from the way whose victim flag is true.

[0022] Referring now to Figure 3, a schematic diagram of messages and transactions between outer and inner caches is shown, according to one embodiment of the present disclosure. In the Figure 3 embodiment, outer cache 310 and inner cache 320 are in an outer and

inner relationship with each other and processor 330. For the sake of clarity, only one representative set is shown in each cache. In one embodiment, outer cache 310 may be the cache 200 of Figures 2A and 2B above.

5 **[0023]** In the Figure 3 example, cache control logic has made true the victim flag of the cache line in way 312. A message 340 may be sent to inner cache 320 to back-invalidate any corresponding cache line within inner cache 320. Let way 324 contain a corresponding cache line to that contained in way 312. Then the cache line in way 324 may be
10 invalidated.

[0024] At a later time, processor 330 may make a request 342 for that recently-invalidated cache line. As that cache line is now invalid within inner cache 320, a second request 344 may be sent to outer cache 310. When the cache control logic of outer cache 310 receives that second
15 request 344, it may determine that that request is for a cache line in a way whose victim flag is true. Cache control logic may respond to this situation by making false the victim flag of way 312, making true the victim flag of another way, for example, way 314 (message 348), and by then sending the requested cache line from way 312 to the inner cache
20 320 (message 346). Since way 314 now has its victim flag true, a message 350 may be sent to inner cache 320 to back-invalidate any cache line within inner cache 320 corresponding to that in way 314. Let way 322 contain a corresponding cache line to that contained in way 314. Then the cache line in way 322 may be invalidated.

25 **[0025]** By making false the victim flag of way 312 and making true the victim flag of way 314, the cache control logic has reacted to a situation where, by the operation of the processor and its associated

caches, a less than optimal selection of the cache line in way 312 as a potential victim was corrected by subsequently selecting the cache line in way 314 as an alternate victim. This process of selecting alternate victims may continue until such time as an actual eviction (or
5 overwriting) takes place on the way whose victim flag is currently true. This process may thus be considered a corrective procedure to the original method of selecting a victim cache line, whether that method is LRU, random, FIFO, or another method.

[0026] Referring now to Figure 4, a schematic diagram of messages
10 and transactions between coherent caches is shown, according to one embodiment of the present disclosure. The Figure 4 embodiment presumes that the caches shown are inclusive. When a particular cache line is requested by processor A 404, a request is sent to L1 cache A 414. If the cache line is not in L1 cache A 414, then a
15 subsequent request is sent to L2 cache A 424. If the cache line is also not in L2 cache A 424, then a request may be made to memory 430: however, the L2 cache A 424 may first snoop other caches in the cache coherency domain to see if the desired cache line is present there.

[0027] If L2 cache B 428 has the desired cache line, in either an
20 exclusive E state or a shared S state, then L2 cache B 428 may signal this to L2 cache A 424 by, among other actions, sending a "cache hit found" HIT 438 signal. L2 cache A 424 may then receive the desired cache line in a shared S state from L2 cache B 428. However, if L2 cache B 428 has the desired cache line in a modified M state, then L2
25 cache B 428 may signal this to L2 cache A 424 by, among other actions, sending a "cache hit modified found" HITM 440 signal. In this situation, L2 cache B 428 needs to invalidate the modified cache line,

both by local invalidation and also by sending a back-invalidate message to all caches inner to L2 cache B 428, before sending the desired but modified cache line to L2 cache A 424. In the Figure 4 embodiment, the inner cache shown is L1 cache B 418, so L2 cache B 428 sends a back-invalidate message to L1 cache B 418. The processing of this back-invalidate message may delay the sending of the modified cache line to L2 cache A 424.

[0028] In one embodiment, if the desired but modified cache line in L2 cache B 428 had the corresponding victim flag made true, then the back-invalidate process to inner caches had been performed at a previous time. Thus in the situation where the desired cache line is not only modified but also has its victim flag true, there may be no need to delay sending the modified cache line over to L2 cache A 424. Hence in this situation there may be a further performance enhancement when utilizing the victim flag.

[0029] Referring now to Figures 5A and 5B, flowcharts of a method of operation of an in-situ victim cache are shown, according to one embodiment of the present disclosure. Figure 5A generally shows a process when responding to a cache line request from a processor core, whereas Figure 5B generally shows a process when responding to a cache line request from a snoop. The Figure 5A process begins at block 510 with the request of a line from the processor core. In decision block 512, it is determined whether the requested line is resident in the cache. If so, the process exits decision block 512 via the YES path and the line is send down to the core in block 514. Then in decision block 516 it is determined whether the victim flag of that line is set true. If not, then the process exits decision block 516 along the NO path and

the process repeats. If, however, the victim flag of that line is true, then the process exits decision block 516 along the YES path and in block 518 that victim flag is cleared (set false).

[0030] If, on the other hand, the requested line was not resident in the cache, then the process exits decision block 512 along the NO path and in block 530 the requested line is retrieved from memory. In block 532 that line may then be immediately forwarded to the processor core. Then in decision block 534 it is determined whether a way in the cache has its invalid flag set true. If so, then the process exits decision block 534 via the YES path and in block 536 that way receives the retrieved line. If not, then the process exits decision block 534 via the NO path. The process maintains a line with its victim flag set true when there is no line with an invalid flag set true, so in block 538 that line is evicted and then in block 536 the corresponding way receives the retrieved line.

[0031] The process enters decision block 520 from either block 518 or block 536. In decision block 520 it is determined whether there exists either a line with its victim flag set to true or its invalid flag set to true. If so, then the process exits along the YES path to completion in block 540. If not then the process exits along the NO path to block 522, where a line in a particular way is chosen to have its victim flag set to true in block 544. In block 542 that chosen line is back invalidated, and then the process completes in block 540.

[0032] In Figure 5B, the process begins with a snoop request in block 550. In decision block 552 it is determined whether the requested line is in the cache. If not, then the process exits decision block 552 via the NO path and the process repeats without further action. If so, then the process exits decision block 552 via the YES path and in decision block

554 it is determined whether the requested line is in a modified state. If not, then the process exits decision block 554 via the NO path and the process repeats without further action. If so, then the process exits decision block 554 via the YES path, and in decision block 556 it is
5 determined whether the requested line has its victim flag set true. If not, then the process exits decision block 556 via the NO path and the requested line is back invalidated in block 560 before writing the line back in block 558. If however, the requested line does have its victim flag set true, then the process exits decision block 556 via the YES
10 path, and the cache line advantageously does not need back invalidation before writing the line back in block 558. In either case the process may then repeat.

[0033] Referring now to Figure 6, a schematic diagram of a microprocessor system is shown, according to one embodiment of the
15 present disclosure. The Figure 6 system may include several processors of which only two, processors 40, 60 are shown for clarity. Processors 40, 60 may include level two caches 42, 62, and other caches (not shown) inner to level two caches 42, 62. The Figure 6 multiprocessor system may have several functions connected via bus interfaces 44, 64,
20 12, 8 with a system bus 6. In one embodiment, system bus 6 may be the front side bus (FSB) utilized with Pentium® class microprocessors manufactured by Intel® Corporation. In other embodiments, other busses may be use, or point-to-point interfaces may be used instead of a bus for memory access. In some embodiments memory controller 34
25 and bus bridge 32 may collectively be referred to as a chipset. In some embodiments, functions of a chipset may be divided among physical chips differently than as shown in the Figure 6 embodiment.

[0034] Memory controller 34 may permit processors 40, 60 to read and write from system memory 10 and from a basic input/output system (BIOS) erasable programmable read-only memory (EPROM) 36. In other embodiments, memory controller 34 may connect via a point-to-point interface to level two cache 42 of processor 40, eliminating bus interfaces 44, 8. In some embodiments BIOS EPROM 36 may utilize flash memory. Memory controller 34 may include a bus interface 8 to permit memory read and write data to be carried to and from bus agents on system bus 6. Memory controller 34 may also connect with a high-performance graphics circuit 38 across a high-performance graphics interface 39. In certain embodiments the high-performance graphics interface 39 may be an advanced graphics port AGP interface. Memory controller 34 may direct read data from system memory 10 to the high-performance graphics circuit 38 across high-performance graphics interface 39.

[0035] Bus bridge 32 may permit data exchanges between system bus 6 and bus 16, which may in some embodiments be a industry standard architecture (ISA) bus or a peripheral component interconnect (PCI) bus. There may be various input/output I/O devices 14 on the bus 16, including in some embodiments low performance graphics controllers, video controllers, and networking controllers. Another bus bridge 18 may in some embodiments be used to permit data exchanges between bus 16 and bus 20. Bus 20 may in some embodiments be a small computer system interface (SCSI) bus, an integrated drive electronics (IDE) bus, or a universal serial bus (USB) bus. Additional I/O devices may be connected with bus 20. These may include keyboard and cursor control devices 22, including mice, audio I/O 24,

communications devices 26, including modems and network interfaces,
and data storage devices 28. Software code 30 may be stored on data
storage device 28. In some embodiments, data storage device 28 may
be a fixed magnetic disk, a floppy disk drive, an optical disk drive, a
5 magneto-optical disk drive, a magnetic tape, or non-volatile memory
including flash memory.

[0036] In the foregoing specification, the invention has been described
with reference to specific exemplary embodiments thereof. It will,
however, be evident that various modifications and changes may be
10 made thereto without departing from the broader spirit and scope of the
invention as set forth in the appended claims. The specification and
drawings are, accordingly, to be regarded in an illustrative rather than a
restrictive sense.